Performance Issues in a Real-Time True Color Data Display

Robert F. Erbacher and Georges G. Grinstein
Institute for Visualization and Perception Research
Department of Computer Science
University of Massachusetts-Lowell
One University Avenue
Lowell, MA 01854

Abstract

We discuss the integration of visualization and supercomputing in a low cost environment. Computational requirements continue to increase dramatically as computational capabilities do. Yet most architectures still separate both processes. The computation is done on one system and the visualization on another. We describe an innovative architecture developed by the Supercomputer Research Center of the Institute for Defense Analysis within which the integration of visualization and supercomputation is realized. Immediate gains are obvious: program visualization, real-time computational steering, and rapid porting of current applications. We describe the issues in porting our experimental visualization software1,2,7 and issues we encountered. We describe limitations and advantages of the hardware/software coupling. We also discuss a proposed extension of that architecture.

1 Introduction

Recently we were asked to implement and port some of our visualization applications to a proprietary platform (called "Terasys") developed by the Supercomputing Research Center (SRC) of the Institute for Defense Analyses. The goal was to implement our routines in such a way as to provide real-time interaction with true color displays of images of at least 512×512 pixels. The Terasys platform provided by SRC is a SIMD architecture which serves as the back end to a Sparc II. An interface card sitting on the S-bus provides the connection between the two components. Rosario and Choudhary have discussed the limitations of current I/O support in massively parallel computers and the impact this can have on the actual performance of the architecture.[8] This paper concentrates on our efforts to perform display updates in real-time when the data to be displayed is retrieved from the Terasys. We emphasize hardware and software issues related mainly to the capabilities of the Sparc II.

2 Exvis

The Exploratory Visualization Environment (Exvis) provides a context within which experiments in the presentation (visualization or sonification) of multiparameter data may be performed; such data can also come from large databases, imagery and categorical

2.1 Iconographics

Exvis incorporates several state-of-the-art techniques for the rendering of scientific data. These techniques are based on the concept of "iconographic" displays. An iconographic display associates an icon—a figure of arbitrary size—with each pixel of the original image. The various visual attributes of an icon are under data control. The exploratory visualization environment which incorporates these iconographic techniques is termed "Exvis" (Exploratory Visualization).

Because most data sets consist of more than a single image, we have developed icons that allow us to map multiple images (parameters) onto a single icon. This mapping ability allows us to merge the images or parameters into a single display which hopefully provides more information than is observable from viewing the individual images side by side. An important concept here is that icons provide more information at the cost of screen resolution (for each pixel in an original image, an iconographic display has an area of pixels to contain each icon).

2.2 The stick-figure icon

The first icon, developed by Pickett[7], is termed the "stick figure" icon. A stick figure icon is formed by connecting line segments at their endpoints according to a set of geometric rules. The angles, lengths, and grayscale values of the line segments can be controlled by data. Two of the data variables, not necessarily independent of those controlling the icon features but hopefully largely independent of each other, control the position of each icon on the display surface. When icons are used to represent imagery data, icon position simply corresponds to pixel position. With sufficient density, the icons form a surface texture display, and structures in the data are revealed as streaks, gradients, or islands of contrasting texture.

2.3 The color icon

The stick figure icon relies primarily on the perception of line orientation to reveal information. Another approach is to use color perception. Color is a very effective way to merge multiple parameters of imagery data. For up to three parameters, the values of the pixels in the separate images control the three color channels of the corresponding pixel in the integrated display. Different mappings of values to color channels and different color models provide a wide range of options. Levkowitz[3,4,5] developed a generalized color icon to integrate multiple images using both color and geometric features. In this icon a rectangle of pixels is used to represent each point in a data set. The color channels of the four corners of the icon may be independently data controlled—providing 12 parameters. The pixels within the icon are interpolated to obtain the final coloring of the icon as a whole. The color icon provides straight color pixel integration when the width and height of the icon are both set to one pixel. Interactive controls allow the user to roam through a space of color models to find a model that provides the most effective presentation of the data.

2.4 Performance of iconographic techniques

Because iconographic techniques tend to be computationally expensive, several seconds may be required to render a single display using these techniques on a serial architecture. This rendering time, combined with the time required to find useful settings for the dozens of variables that control the rendering transformation, makes the process of exploring the possible representations of a data base extremely time-consuming. Moreover, because many recent databases are very large and have hundreds of parameters, the task of fully exploring such data bases appears overwhelming. Consequently, an interest in supercomputer implementations of these techniques has developed. It is hoped that these implementations will speed up the rendering time and substantially reduce the time required to fully explore a database. Smith and Grinstein previously described a supercomputer implementation of the stick figure icon.[9] In this paper we will discuss implementation details of the supercomputer version of the color icon and briefly outline the benefits and performance of this system.

3 Hardware Architecture

The hardware provided to us for this project is the "Terasys" system developed by the Supercomputing Research Center.[6,10] This system provides a bit-serial SIMD architecture in a linear array organization. The system we were provided with currently contains 4K processors, although up to 32K

processors are supported. Each processor operates on a local 2K-bit memory, which is allocated to variables whose lengths may be specified. The Terasys also provides a parallel prefix network which improves the performance of some important operations. A SUN Sparc II plays hosts to the Terasys hardware, which resides on the S-bus. The performance of the S-bus was found to be a limiting factor for the real-time graphical displays.

When choosing the graphics engine for this project, we ensured that the board selected would allow us to access the framebuffer directly. We did this because we felt that operating through an X server would be too limiting. While the chosen board, a RasterFLEX-HR, did not provide library routines to access the board directly, it did provide an example for memory mapping the framebuffer and directing data to the board in that manner.

4 Software Architecture

The work we have performed on the Terasys consisted of two main parts. The first part dealt with showing the applicability of the Terasys hardware to perform basic graphics operations. The second part of the project dealt with porting a section of the visualization environment developed by the University of Massachusetts-Lowell's Institute for Visualization and Perception Research.

During the first part of the project a library of graphics operations was built to assist in the inclusion of graphical displays into parallel programs. The library was designed with the hope that individuals without graphics experience could add a certain amount of visual displays to their programs. In addition, it also separated out the code that is dependent on the graphics hardware. Consequently, if a different graphics board is used or a better mechanism is provided for performing the display of data then only the code in the library should need to be changed. Otherwise the code in every demo/application would have to be changed to include the modified code.

This portion of the project also incorporated the implementation of several graphics algorithms on the Terasys hardware. These algorithms consisted of: matrix transposition, matrix convolution, matrix multiplication, and sorting. These algorithms were implemented such that every step of the computation is displayed graphically such that the user can watch the programs execute dynamically. This is generally termed program animation. Program animation is generally used to help in understanding how an algorithm operates and requires that the display update dynamically such that the user gains insight into the flow of data. Additionally, it is usually desirable to display several variables of the algorithm to better assist the user in understanding the algorithm. Consequently, the effectiveness of a program animation system can be highly dependent on the performance of the system. In our case, since we are performing all computation on a supercomputer the dependent factor is the performance of the display subsystem.

The second portion of the project was of particular importance since it required the implementation of an interactive application. This part of the project consisted of implementing the color icon described previously. The desire was to provide an environment which allowed real-time manipulation of parameters such that informative displays could be generated very quickly. It was also hoped that being able to manipulate controls in real-time would give users an intuitive understanding of the effects of the different controls—resulting in better use of those controls.

During both phases of this project the rate at which images could be calculated and displayed was of particular interest. Since we were unable to generate displays in true real-time it was important to identify where the bottlenecks were such that alternatives could be considered. While we were able to improve the throughput by bypassing X and writing data directly to display memory the display time

is still the limiting performance factor. A best case scenario would provide a frame buffer directly on the Terasys hardware; which should greatly improve performance.

5 Porting Exvis

Exvis has been running on a wide variety of platforms, ranging from the Macintosh platform to a variety of UNIX environments. Smith and Grinstein[9] discuss the port to the Connection Machine and have shown the importance of tightly coupling supercomputing, visualization and data exploration. However the cost of a CM2 is beyond most users. SRC, with Terasys, have shown that supercomputing hardware can be developed at costs near co-processing boards. We ported portions of the Exvis-II interface, including the color icon developed by Haim Levkowitz, to the Terasys hardware. This implementation was done as efficiently as possible. The desire for real-time interaction with this interface forced us to explore in detail the effects of the S-bus on the performance of the system.

For a 320×320 pixel image, 64×64 icons, we can display ~5 frames per second using the Terasys hardware. It is important to note that only ~.05 of a second are spent on the actual calculation time. The remaining ~.15 of a second are spent performing the actual display of the image. Increasing the number of processors available on the system will allow us to increase the size of the generated image. The computation time will remain constant, characteristic of all SIMD hardware, but the display time will increase linearly with the increase in processors (assuming the algorithm remains unchanged). This shows the importance of a high throughput bus.

6 Performance Tests

In order to provide better controlled tests for performance analyses, we also obtained information on how to read data directly from the Terasys and on the form of the retrieved data. This information enabled us to break down the display functions into different routines. These routines could then be implemented with different methods in order to find the best implementation and also to provide costs for the different portions of the overall display facilities.

6.1 Throughput measurements

The testing we performed consisted initially of tests designed to dump data to the graphics board as quickly as possible in order to allow us to examine the feasibility of real-time displays. We performed many tests during this portion of the project. For this paper the tests of primary importance are the following:

A. This test performed a memory copy from a region of standard memory to the video frame buffer. The frame buffer was memory-mapped into the user address space. This test examined the raw throughput of the frame buffer. This test was not very useful because we were simply dumping data, not aligning it to fit into a window. Thus, the display appeared as a few horizontal lines across the entire display.

B. This test was similar to test A in that we were writing directly to the frame buffer. It differed in that we were doing the writing pixel by pixel, rather than as a chunk with memory copy. This test also performed the required operations to center the data within a window. This test, together with Test A, give insights into realistic frame buffer throughput rates and also the amount of time spent on organizing pointers to write data into the frame buffer correctly. The data for this test were also taken from memory. The actual implementation of the direct frame buffer display routine writes each row of data into the framebuffer at one time with a memory copy. Thus, the actual performance is somewhat increased.

C. Time required to display an Exvis image (without computation time) using X. This includes reading data from the Terasys and writing data to the framebuffer. The user was assumed to be local to the XServer and shared memory pixmaps were used for this test.

D. Time required to display an Exvis image (without computation time) bypassing X and writing directly to the frame buffer. This includes reading data from the Terasys and writing data to the framebuffer.

E. Computation time required to generate an Exvis image. These results were provided for comparison with the display times in order to provide an understanding of how much of a bottleneck the displaying of data is in comparison to the computation of data.

The following table shows the results of these tests for 256^2 and 512^2 pixel images. While these image sizes were used for test purposes, images of 320^2 are used for general use as they provide the most useful image size/performance ratio. The table shows the number of frames achievable per second for the given test.

	Frame Rate (frames/sec.)	
Tests	256×256 pixel image	512×512 pixel image
A	43.21	10.81
B	30.97	7.74
C	4.43	1.13
D	7.02	1.81
E	50	16.13

It is important to note that for this system we have a true color display board which requires 32 bits for each pixel value. Thus, each display requires four times as much data as an 8-bit display provided with most general purpose workstations.

6.2 Analysis of throughput

The tests showed that for a 512×512 image we should be able to display about 7.5 frames/second with the data coming from memory. This is a bit on the slow side, but it would very likely be usable for close to real-time interaction. Unfortunately, the cost of computing the display data and the cost of retrieving that data from the Terasys hardware must also be taken into account. Because the data to be displayed isn't coming from memory but rather from another device on the S-bus, the actual time required merely to copy data from one device to another would allow only about 4 frames/second, assuming the Terasys hardware could perform reads at that rate. This is generally too slow to permit interaction in real-time, particularly after data calculation time is incorporated. The main reason for this performance problem is a lack of bandwidth across the S-bus. The S-bus incorporated in the Sparc II is incapable of moving the amount of data required for large true color images.

As it turns out, the Terasys adds enormous overhead to the costs already given. These costs fall into two classes: read latency and the cost of decoding the data retrieved from the hardware into a usable form. After these costs are added we were generating updates at about one frame per second on images of 512×512 pixels.

6.3 Issues of high performance display updates

Several other issues concerning X Servers in general and the RasterFLEX X server in particular emerged during this testing. While attempting to write directly to the framebuffer, it is possible for the X Server to interfere with the writes, or vice versa. Problems can occur when an application which bypasses the X Server is performing an update and the X Server makes modifications to variables in a graphics board register. Another problem is that the X Server provided by RasterFLEX uses XImages which are padded. In other words the number of bytes per line is not equal to the width of the image times the number of bytes per pixel. This makes it much more difficult to use XImages, which apply nicely to this type of application when we can t access the framebuffer directly.

Adjusting for the padding in the XImages causes a significant degradation in performance since we would otherwise be able to transfer result data directly from the Terasys hardware into the XImage. To deal with this problem we instead used shared memory pixmaps. Shared memory pixmaps offer most of the capabilities of both XImages and Pixmaps, except for their being slower than normal pixmaps, without the problem of padding observed with XImages. This allowed us to get a very efficient implementation while using the XServer.

While attempting to generating a high throughput system we decided to attempt bypassing the X Server all-together. The main problem we found here is that of the 32 bits provided per pixel only 24 of them are actually used for the color of the pixel. The remaining 8 bits are used to identify the overlay factor of the pixel. Setting the plane mask register of the graphics board to not modify the overlay mask does not work satisfactorily. First, this plane mask must be set for every write to the framebuffer which greatly increases the time required to display the entire image. Second, this does not ensure that the overlay mask will not be modified since movement of the mouse cursor will modify the plane mask (a consequence of using a multi-tasking operating system). Since the overlay mask is the same for the entire window and should not be modified for our purposes we found it more effective to read the overlay mask during window creation and to place this mask in the overlay bits of all the data to be sent to the framebuffer. Since we were using a SIMD architecture modifying the resulting data in this manner was trivial. Since all data transfers over the S-bus are 32 bits anyway we are not losing any performance by making the bits in the overlay mask meaningful. Writing directly to the framebuffer as opposed to using the X Server provided about a 20% increase in the over all display time for a 320×320 pixel Exvis image (including image calculation time).

Although the Sparc II platform did not live up to our expectations for performance, it did allow us to successfully and easily pinpoint the real-time performance bottlenecks for this particular research project.

7 Frame Buffer

The greatest bottleneck at this point comes from sending data between the SUN and the Terasys hardware itself. Most graphical applications will compute the desired image on the SUN itself. The results are then sent over the S-bus to the framebuffer. In our case, we are computing the image on the Terasys hardware. Since the data is not stored on the Terasys hardware in a form directly usable by the framebuffer the data is then returned to the Sparc processor for final adjustment. Finally, the data is sent over the S-bus a second time to the framebuffer.

The S-bus has been shown to be too limiting for the types of applications discussed here. As mentioned, all data which is to be displayed must cross this bottleneck twice under our system. Once to bring the data from the Terasys hardware to the CPU and once to send the data to the framebuffer from the CPU. With 24-bit data the 32MB/s throughput is just too limiting and is out of date. New

buses in the PC realm, the PCI bus in particular, are achieving bursts of up to 132 MB/s. Since the PCI bus is designed to be processor independent it provides the additional benefit of allowing the hardware to be used on multiple platforms.

8 Conclusions and Future Work

We have shown that the Sun bus is greatly reducing the throughput of graphical display routines on this hardware and have suggested methods for alleviating this problem.

As we have shown, the Sparc bus is incapable of maintaining the throughput necessary for near real-time interaction. Using the image calculation times shown above for a 320×320 pixel image and assuming we want to generate 15 frames/second we will need a bus capable of 24.58 MB/sec. Reducing our requirements to 10 frames/sec will require a bus capable of 8.19 MB/sec. This is actual throughput needed by the bus for data (after bus protocol overhead is taken into account). Increasing the desired resolution to 512×512 will increase the needed bandwidth to 62.91 MB/sec. and 20.97 MB/sec. respectively. This is somewhat unrealistic as reducing the computation time will also help alleviate the throughput needed across the I/O Bus and will actually be required for the desired frame rate.

In addition, the bus must be able to handle double the bandwidth just specified since the data must actually cross the bus twice. Once to bring the data from the Terasys hardware to the CPU and once to finally send the data from the CPU to the framebuffer. This need could be alleviated but at a cost. First, a bus mastering controller would need to be provided on the Terasys end of the S-bus. A bus mastering controller would allow the Terasys adapter card to send data directly to the framebuffer. Unfortunately, since the data, as stored on the Terasys, is not in the correct format for the framebuffer additional computational capability would need to be added to the Terasys interface card to translate the data into a usable form. This would greatly increase the cost of the Terasys system.

While we are currently only generating ~5 frames/sec. (320×320 pixel images), which is nowhere near real-time, we have found that this is sufficient for test purposes. Parameters can be changed interactively with the display updating dynamically with minimal interference from the lack of throughput. This dynamic updating aids in understanding how the different parameters affect a given display and helps to develop an intuitive understanding of what parameters to change to improve the effectiveness of a display. Additional throughput would smooth the display—reducing the jerky appearance.

Lastly, it is important to mention that the algorithm used to generate the iconographic displays was optimized for the display portion of the code. The computation could have been performed much more quickly but this would have made the display time of the resulting image prohibitive.

9 Acknowledgments

The work described here was funded by a grant from the Super Computing Research of the Institute for Defense Analyses.

References

1. G. G. Grinstein, R. M. Pickett, and M. G. Williams, "EXVIS: An Exploratory Visualization Environment," *Graphics Interface 89*, London, Ontario, 1989.
2. G. G. Grinstein and S. Smith, "The Perceptualization of Scientific Data," *Proceedings of SPIE 90*, February 1990.

3. H. Levkowitz and G. T. Herman, "Towards a Uniform Lightness, Hue, and Saturation Color Model," *Proceedings of the SPIE Conference Image Processing, Analysis, Measurement, and Quality*, pp. 215-222, 1988.
4. H. Levkowitz, "Color Icons: Merging Color and Texture Perception for Integrated Visualization of Multiple Parameters," *Proceedings of the Visualization 91 Conference*, IEEE Computer Society Press, San Diego, CA, October 22-25, 1991.
5. H. Levkowitz and G. T. Herman, "GLHS: A Generalized Lightness, Hue, and Saturation Color Model," *CVGIP: Graphical Models and Image Processing*, Vol. 55, No. 4, July, pp. 271-285, 1993.
6. Jennifer Marsh and Mark Norder, "Pim Chip Specification," Technical Report SRC-TR-93-088, Super Computing Research Center, Institute for Defense Analyses, Bowie, MD, 1993.
7. R. M. Pickett and G. G. Grinstein, "Iconographic Displays for Visualizing Multidimensional Data," *Proceedings of the 1988 IEEE Conference on Systems, Man and Cybernetics, Beijing and Shenyang*, People's Republic of China, 1988.
8. J. Rosario and A. Choudhary, "High Performance I/O for Massively Parallel Computers," *IEEE Computer*, March 1994, pp. 59-68.
9. S. Smith, G. Grinstein, and R. D. Bergeron, "Interactive Data Exploration with a Supercomputer," *Proceedings of the IEEE Visualization 91 Conference*, IEEE Computer Society Press, San Diego, CA, 1991.
10. H. Douglas Sweely, "Terasys Demonstration Hardware Manual," SRC No. 101.93, Super Computing Research Center, Institute for Defense Analyses, Bowie, MD, 1993.